Music Tech Workshop

Open Sound Control (OSC) & SuperCollider

by Antoine (@notkaramel)

Sections

Why use OSC? What is SuperCollider

---

1. Install ~ 2. Getting Started
3. SC Basics ~ 4. Synth & SynthDef
5. Signal Processing ~ 6. Envelope
7. Output & Panning ~ 8. Server, OSC
9. Extras

🌟 **Game time!** 🌟

Prerequisite knowledge

It is recommended that you have some background knowledge on

- **Object Oriented Programming (OOP)**
- **Signal processing**
- **The client-server architecture**
- **Network protocol (UDP, HTTP, TCP)**

Links

Project Presentation:

https://notkaramel.github.io/MUMT306/

Project Source Code:

https://github.com/notkaramel/MUMT306

What is Open Sound Control?

What is Open Sound Control?

Why use OSC?

What is Open Sound Control?

Why use OSC?

Why not use MIDI???

What is Open Sound Control?

Why use OSC?

Why not use MIDI???

What is SuperCollider?

"What is Open Sound Control?"

"What is Open Sound Control?"

Open Sound Control (OSC) is a communication protocol for *computers*, *sound synthesizers*, and other *multimedia devices* that is *optimized for modern networking*.

OSC Proposal at ICMC, 1997

"What is Open Sound Control?"

Open Sound Control (OSC) is a communication protocol for *computers*, *sound synthesizers*, and other *multimedia devices* that is *optimized for modern networking*.

OSC Proposal at ICMC, 1997

OSC is designed as a *highly accurate*, *low latency*, *lightweight*, and *flexible* method of communication for use in *realtime musical performance*.

OpenSourceControl website

"Why use OSC?"
"Why not use MIDI?"

"Why use OSC?"
"Why not use MIDI?"

In a way, **OSC** is a *more flexible alternative* to **MIDI**.

"Why use OSC?"
"Why not use MIDI?"

In a way, **OSC** is a *more flexible alternative* to **MIDI**.

MIDI is a protocol standard made for *hardware synthesizer* with *electrical circuit limitations*.
Meanwhile, **OSC** travels through the internet via UDP (or User Datagram Protocol).

"What is SuperCollider?"

"What is SuperCollider?"

In this workshop, we will **not** go into the **encoding of OSC** and all the nitty gritty details of it.

"What is SuperCollider?"

In this workshop, we will **not** go into the **encoding of OSC** and all the nitty gritty details of it.

We will use a program called **SuperCollier** to use and interact directly with OSC and create an application using it.

Step 1: Get SuperCollider

Download SuperCollider (Windows, Linux, MacOS):
https://supercollider.github.io/downloads.html

Step 1: Get SuperCollider

Download SuperCollider (Windows, Linux, MacOS):
https://supercollider.github.io/downloads.html

For the *command-line enthusiasts*

```
choco install supercollider          # Windows
apt install supercollider            # Debian-based
pacman -S supercollider              # Arch-based
brew install --cask supercollider    # MacOS
```

Step 2

Getting started with the SuperCollider IDE

Let's make a `Hello World` program.

```
"Hello World!".postln;
```

Let's make a `Hello World` program.

```
"Hello World!".postln;
```

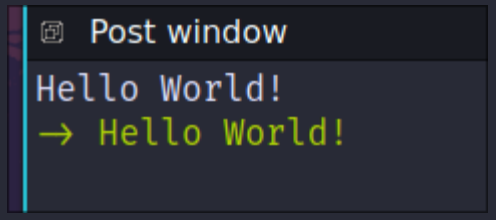
Press `Ctrl Enter` or `Cmd Enter`

Let's make a `Hello World` program.

```
"Hello World!".postln;
```

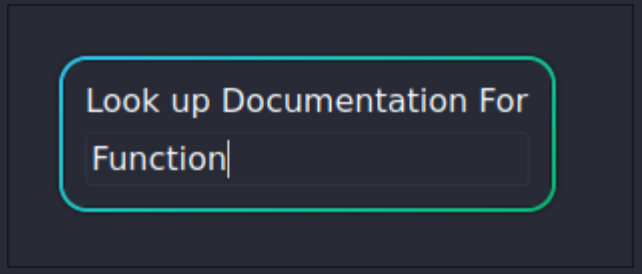
Press `Ctrl Enter` or `Cmd Enter`

You should get this on the post window.



Documentation

SuperCollider has a very extensive documentation. You can access it by pressing `Ctrl Shift D` or `Cmd Shift D`



Documentation

You can also access the documentation of a function by `Ctrl D` or `Cmd D` with the cursor on the code.

Documentation

You can play with the example codes on the documentation with `Ctrl Enter` or `Cmd Enter`

Examples

```
// create an audio-rate sine wave at 200 Hz,
// starting with phase 0 and an amplitude of 0.5
{ SinOsc.ar(200, 0, 0.5) }.play;

// modulate the frequency with an exponential ramp
{ SinOsc.ar(XLine.kr(2000, 200), 0, 0.5) }.play;

// more complex frequency modulation
{ SinOsc.ar(SinOsc.ar(XLine.kr(1, 1000, 9), 0, 200, 800), 0, 0.25) }.play;

// phase modulation (see also PMOsc)
{ SinOsc.ar(800, SinOsc.ar(XLine.kr(1, 1000, 9), 0, 2pi), 0.25) }.play;
```

Step 3: SC Basics

Let's start with making a sinusoid wave!

3.1. Create a `SinOsc` object with the params:

```
SinOsc.ar(440);
```

You can always check the documentation:

Class Methods

`SinOsc.`**`ar`**`(freq: 440.0, phase: 0.0, mul: 1.0, add: 0.0)`

`SinOsc.`**`kr`**`(freq: 440.0, phase: 0.0, mul: 1.0, add: 0.0)`

Arguments:

freq	Frequency in Hertz. Sampled at audio-rate.
phase	Phase in radians. Sampled at audio-rate.

> **NOTE:** phase values should be within the range +-8pi. If your phase values are larger then simply use `.mod(2pi)` to wrap them.

mul	Output will be multiplied by this value.
add	This value will be added to the output.

3.1. Create a `SinOsc` object with the params:

```
SinOsc.ar(440);
```

Note: Unspecified parameter(s) will use the default value(s)!!

3.2. Wrap the oscillator with curly brackets.
You now have a `Function`

```
{ SinOsc.ar(440) };
```

3.3. You can add `.play` to play the `Function` to make it play the sound!

```
{ SinOsc.ar(440) }.play;
```

3.4. Press `Ctrl Enter` or `Cmd Enter` with the cursor on the line to play the sine wave

```
{ SinOsc.ar(440) }.play;
```

It's a pure A4 signal!

3.4. Press `Ctrl Enter` or `Cmd Enter` with the cursor on the line to play the sine wave

```
{ SinOsc.ar(440) }.play;
```

It's a pure A4 signal!

Oh and, `Ctrl B` / `Cmd B` to start the `scsynth` local server!

3.5. Press `Ctrl .` or `Cmd .` to stop the audio playing

```
{ SinOsc.ar(440) }.play;
```

3.6. Variables

3.6. Variables

Single character : `a = { ... }`

not recommended because there are reserved/default variable (`s` for server)

3.6. Variables

Single character : `a = { ... }`

not recommended because there are reserved/default variable (`s` for server)

Long word

```
~myThing = ...
or
var myThingy2 = ...
```

can follow any naming convention, very flexible

Step 4

Synth & SynthDef

Remember OOP?

SynthDef

SynthDef

```
1      // Create a new Synth Definition
2      SynthDef.new(\name, {Function});
3
4      // Add (Sending) the SynthDef to the server
5      SynthDef.new(\name, {Function}).add;
```

SynthDef

```
1    // Create a new Synth Definition
2    SynthDef.new(\name, {Function});
3
4    // Add (Sending) the SynthDef to the server
5    SynthDef.new(\name, {Function}).add;
```

SynthDef - Example

SynthDef - Example

```
1     // Create a new Synth Definition
2     SynthDef.new(\Sinusoid, {
3         var freq = 440; // A4
4         var sinFunc = SinOsc.ar(freq);
5         Out.ar(0, sinFunc);
6     }).add;
```

SynthDef - Example

```
1      // Create a new Synth Definition
2      SynthDef.new(\Sinusoid, {
3          var freq = 440; // A4
4          var sinFunc = SinOsc.ar(freq);
5          Out.ar(0, sinFunc);
6      }).add;
```

SynthDef - Example

```
1       // Create a new Synth Definition
2       SynthDef.new(\Sinusoid, {
3           var freq = 440; // A4
4           var sinFunc = SinOsc.ar(freq);
5           Out.ar(0, sinFunc);
6       }).add;
```

SynthDef - Example

```
1       // Create a new Synth Definition
2       SynthDef.new(\Sinusoid, {
3           var freq = 440; // A4
4           var sinFunc = SinOsc.ar(freq);
5           Out.ar(0, sinFunc);
6       }).add;
```

SynthDef - Example

```
1       // Create a new Synth Definition
2       SynthDef.new(\Sinusoid, {
3           var freq = 440; // A4
4           var sinFunc = SinOsc.ar(freq);
5           Out.ar(0, sinFunc);
6       }).add;
```

SynthDef - Example

```
1      // Create a new Synth Definition
2      SynthDef.new(\Sinusoid, {
3          var freq = 440; // A4
4          var sinFunc = SinOsc.ar(freq);
5          Out.ar(0, sinFunc);
6      }).add;
```

We will learn about `Out` function soon!

SynthDef - Example

To run a large section of code, you can add parenthesis at the beginning and the end of the block to run it!

SynthDef - Example

To run a large section of code, you can add parenthesis at the beginning and the end of the block to run it!

```
(
    // Create a new Synth Definition
    SynthDef.new(\Sinusoid, {
        var freq = 440; // A4
        var sinFunc = SinOsc.ar(freq);
        Out.ar(0, sinFunc);
    }).add;
)
```

SynthDef - Example

To run a large section of code, you can add parenthesis at the beginning and the end of the block to run it!

```
(
    // Create a new Synth Definition
    SynthDef.new(\Sinusoid, {
        var freq = 440; // A4
        var sinFunc = SinOsc.ar(freq);
        Out.ar(0, sinFunc);
    }).add;
)
```

Synth - Example

Synth - Example

```
1 x = Synth(\Sinusoid);
2
3 // free the UGen when you're done
4 x.free; // or `Ctrl .` / `Cmd .`
```

Synth - Example

```
1 x = Synth(\Sinusoid);
2
3 // free the UGen when you're done
4 x.free; // or `Ctrl .` / `Cmd .`
```

Synth - Example

```
1 x = Synth(\Sinusoid);
2
3 // free the UGen when you're done
4 x.free; // or `Ctrl .` / `Cmd .`
```

Notice the status bar!

SynthDef - Arguments

To add argument(s) to your SynthDef, there are two ways to do it.

SynthDef - Arguments

To add argument(s) to your SynthDef, there are two ways to do it.

#1: Use `arg` for variable(s)

```
1 (
2     // Create a new Synth Definition
3     SynthDef.new(\Sinusoid, {
4         arg freq = 440; // A4
5         var sinFunc = SinOsc.ar(freq)
6         Out.ar(0, sinFunc);
7     }).add;
8 )
```

SynthDef - Arguments

To add argument(s) to your SynthDef, there are two ways to do it.

#2: Use the pipe symbol `| variable |`

```
(
    // Create a new Synth Definition
    SynthDef.new(\Sinusoid, {
        | freq = 440 | // A4
        var sinFunc = SinOsc.ar(freq)
        Out.ar(0, sinFunc);
    }).add;
)
```

Synth - with Arguments

Synth - with Arguments

```
1 x = Synth(\Sinusoid, [\freq, 400]);
2 x.set(\freq, 600);
3
4 // Having multiple arguments?
5 x = Synth(\OtherSinusoid, [\freq, 400, \amp, 0.4]);
```

Synth - with Arguments

```
1 x = Synth(\Sinusoid, [\freq, 400]);
2 x.set(\freq, 600);
3
4 // Having multiple arguments?
5 x = Synth(\OtherSinusoid, [\freq, 400, \amp, 0.4]);
```

Synth - with Arguments

```
1 x = Synth(\Sinusoid, [\freq, 400]);
2 x.set(\freq, 600);
3
4 // Having multiple arguments?
5 x = Synth(\OtherSinusoid, [\freq, 400, \amp, 0.4]);
```

🎉 *Congrats! You've mastered the basics of SuperCollider IDE!*

Step 5

Signal Processing Basics

5.1. Basic functions

Sine Wave: $x(t) = A\sin(2\pi ft + \phi)$

Pulse Train: $x(t) = \sum_{n=-\infty}^{\infty} A \cdot \text{rect}\left(\frac{t - nT}{T_p}\right)$

Saw Tooth: $x(t) = \frac{A}{T}t - \left\lfloor \frac{A}{T}t \right\rfloor$

5.1. Basic functions

Sine Wave: $x(t) = A\sin(2\pi f t + \phi)$

Pulse Train: $x(t) = \sum_{n=-\infty}^{\infty} A \cdot \mathrm{rect}\left(\frac{t - nT}{T_p}\right)$

Saw Tooth: $x(t) = \frac{A}{T}t - \left\lfloor \frac{A}{T}t \right\rfloor$

How can you implement those in SuperCollider??

5.1. Basic functions

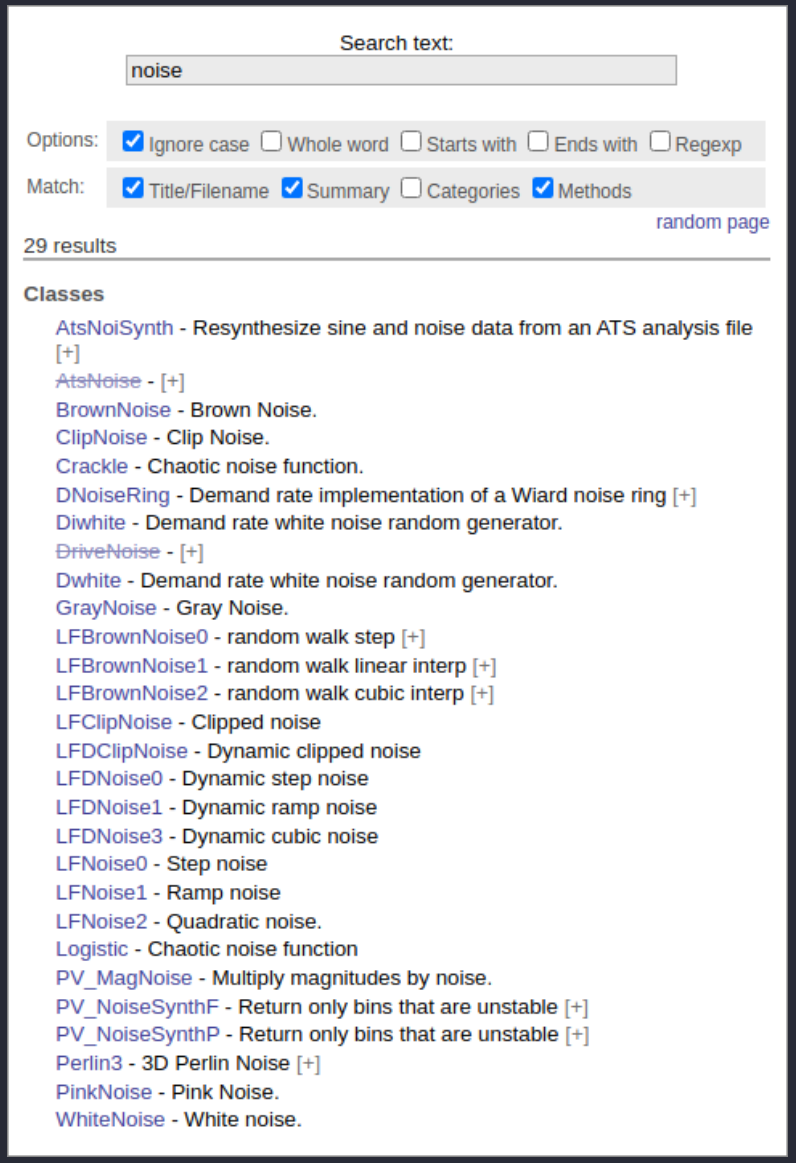
in SuperCollider

Sine Wave: $x = SinOsc.ar(f, \phi, Amp)$

Pulse Train: $x = Pulse.ar(f, width, Amp)$

Saw Tooth: $x = Saw.ar(f, Amp)$

Noises in SuperCollider



Noises in SuperCollider

Noises in SuperCollider

```
1  // Brown Noise
2  BrownNoise.ar(mul: 1.0, add: 0.0);
3
4  // White Noise
5  WhiteNoise.ar(mul: 1.0, add: 0.0);
6
7  // Step Noise
8  LFDNoise0.ar(freq: 500.0, mul: 1.0, add:
9
10 // Pink Noise
11 Pink.ar(mul: 1.0, add: 0.0);
```

5.2. Signal Manipulation

Amplitude Modulation (AM)

$$X_{carrier} = M \sin 2\pi f_{carrier} t$$

$$x(t) = A \cdot X_{carrier} \sin\left(2\pi f t + \theta\right)$$

Frequency Modulation (FM)

$$X_{carrier} = M \sin 2\pi f_{carrier} t$$

$$x(t) = A \sin\left(2\pi f \cdot X_{carrier} t + \theta\right)$$

5.2. Signal Manipulation

Additive & Subtractive Synthesis

$$x_1(t) = A_1 \sin\left(2\pi f_1 t + \theta_1\right)$$
$$x_2(t) = A_2 \sin\left(2\pi f_2 t + \theta_2\right)$$
$$x_{out}(t) = x_1(t) \pm x_2(t)$$

5.2. Signal Manipulation

Additive & Subtractive Synthesis

$$x_1(t) = A_1 \sin\left(2\pi f_1 t + \theta_1\right)$$
$$x_2(t) = A_2 \sin\left(2\pi f_2 t + \theta_2\right)$$
$$x_{out}(t) = x_1(t) \pm x_2(t)$$

```
{
    var freq = 400, amp = 1;
    SinOsc.ar(freq, 0, amp)
    + SinOsc.ar(freq*2, 0, amp/2)
    + SinOsc.ar(freq*4, 0, amp/4)
}.play;
```

5.3. Scopes

SuperCollider has built-in scopes to help us visualize our signals

```
1  // s is the server variable.
2  // Here we want to use the time scope
3  s.scope;
4
5  // There is also a Spectrum Analyzer
6  FreqScope.new(width: 522, height: 300, busNum: 0,
7              scopeColor, bgColor, server)
8
9  // Create a FreqScope for Bus 0 on our current server, 500×4
10 FreqScope(500, 400, 0, s);
```

5.3. Scopes

SuperCollider has built-in scopes to help us visualize our signals

```
1  // s is the server variable.
2  // Here we want to use the time scope
3  s.scope;
4
5  // There is also a Spectrum Analyzer
6  FreqScope.new(width: 522, height: 300, busNum: 0,
7              scopeColor, bgColor, server)
8
9  // Create a FreqScope for Bus 0 on our current server, 500×4
10 FreqScope(500, 400, 0, s);
```

Step 6

Let's make a note!

with Envelope 💌

Envelope function: Env

In essence, an envelope is a function that changes the amplitude of a sound over time. It puts the sound in an "envelope" of sorts, hence the name.

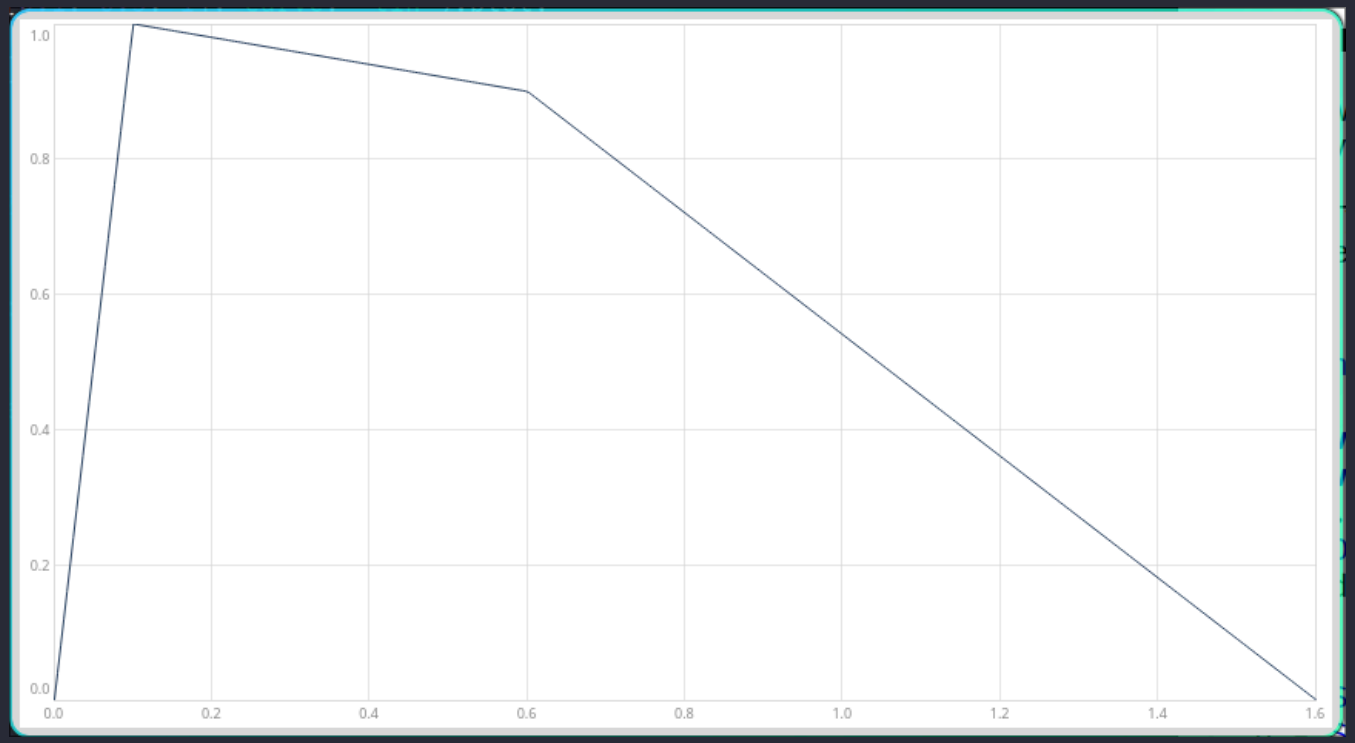
Envelope function: Env

In essence, an envelope is a function that changes the amplitude of a sound over time. It puts the sound in an "envelope" of sorts, hence the name.

There are many ways to create an envelope in SuperCollider.

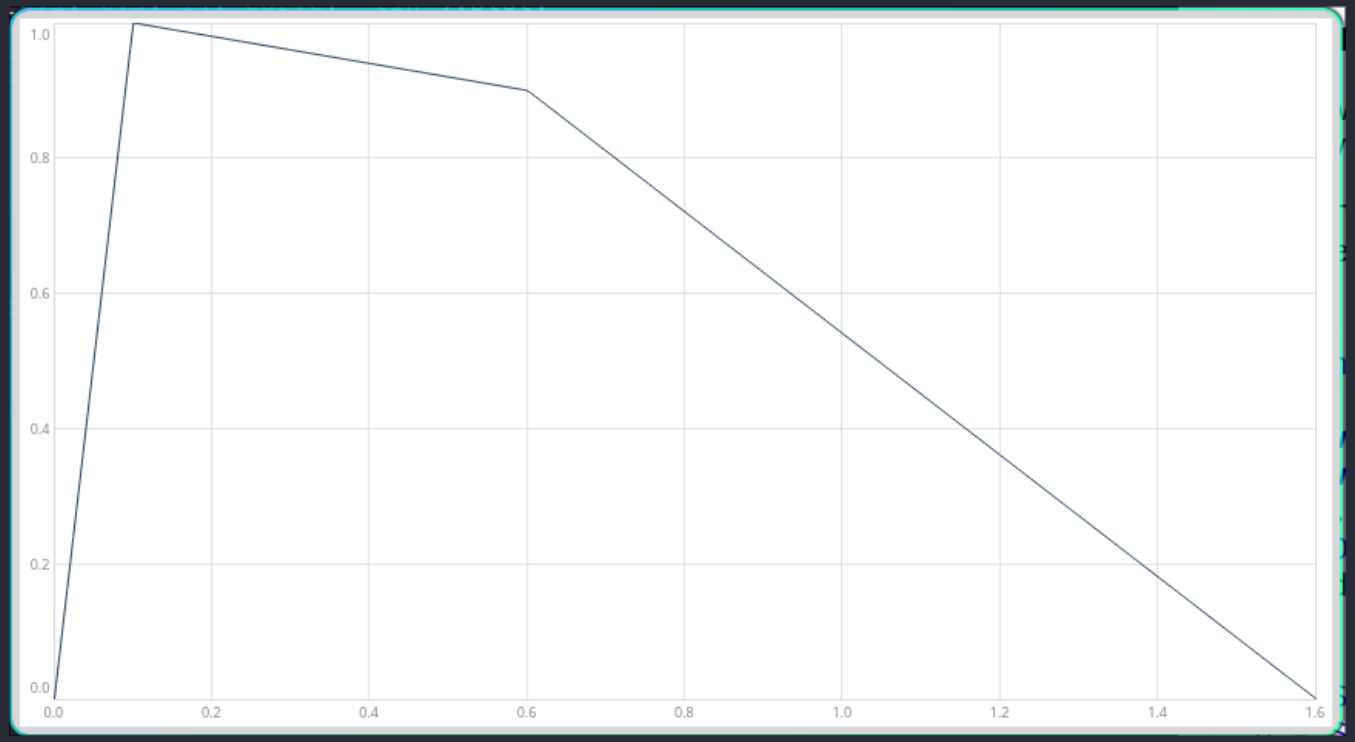
Envelope function: Env

```
1 Env.new(
2     levels: [0, 1, 0.9, 0],
3     times: [0.1, 0.5, 1],
4     curve: 'lin'
5 ).plot;
```



Envelope function: Env

```
1 Env.new(
2     levels: [0, 1, 0.9, 0],
3     times: [0.1, 0.5, 1],
4     curve: 'lin'
5 ).plot;
```



Using Envelope function: `EnvGen`

The `EnvGen` function is used to generate an envelope.

You can think `Env` is the *blueprint* or *specifications* of the envelope, while `EnvGen` is the *instance* that you can use.

Env example

```
~envFunc = Env.perc(0.1, 0.7);
```

```
Env.perc(attackTime: 0.01, releaseTime: 1.0, level: 1.0, curve: -4.0)
```

Creates a new envelope specification which (usually) has a percussive shape.

Arguments:

attackTime	the duration of the attack portion.
releaseTime	the duration of the release portion.
level	the peak level of the envelope.
curve	the curvature of the envelope.

This is also called an **Attack-Release** envelope

Types of envelope

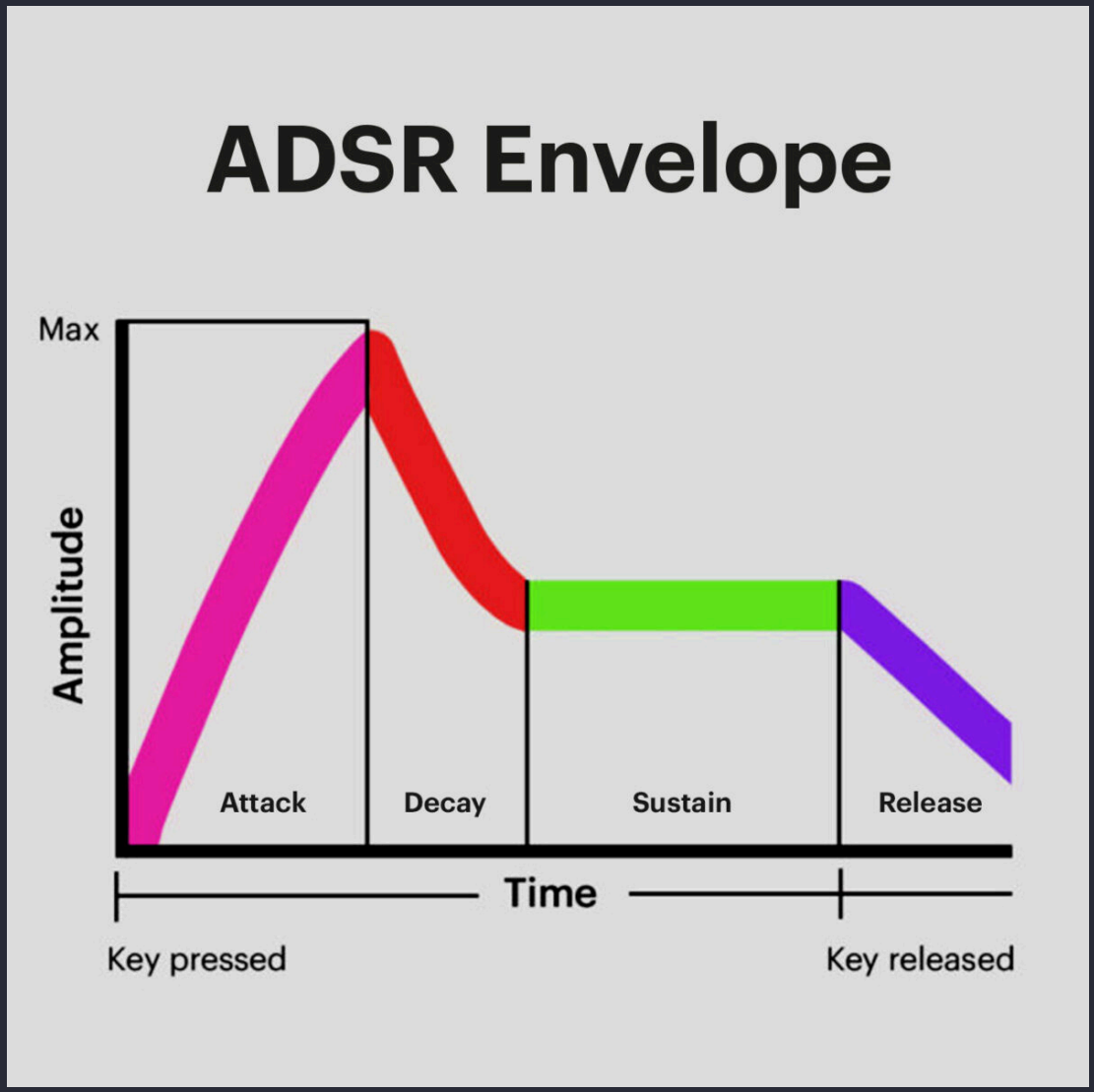
For more types of envelope, you can check the **SuperCollider documentation**.

Additionally, you can checkout the basics and types of envelope at https://thewolfsound.com/envelopes/

Types of envelope: ADSR

ADSR is the most common type of envelope, commonly seen in a lot of *digital audio workstation (DAW)*

```
Env.adsr(attackTime: 0.01, decayTime: 0.3, sustainLevel: 0.5,
  releaseTime: 1.0, peakLevel: 1.0, curve: -4.0, bias: 0.0)
```



https://blog.native-instruments.com/adsr-explained/

Using Envelope function: EnvGen

```
EnvGen.ar(
    envelope,
    gate: 1.0,
    levelScale: 1.0,
    levelBias: 0.0,
    timeScale: 1.0,
    doneAction: 0
);
```

For the scope of this workshop, we will only look into the envelope function and the `doneAction` parameter.

`doneAction` parameter

```
doneAction: Done.freeSelf;
```

This parameter tells the server what to do when the envelope is done.

`doneAction` parameter

```
doneAction: Done.freeSelf;
```

This parameter tells the server what to do when the envelope is done.

`Done.freeSelf` means that the "note" will be freed when the envelope is done.

`doneAction` parameter

```
doneAction: Done.freeSelf;
```

This parameter tells the server what to do when the envelope is done.

`Done.freeSelf` means that the "note" will be freed when the envelope is done.

This concept in SuperCollider is called **nodes (node objects)** on the server.

EnvGen example

```
1 ~envFunc = Env.perc(0.1, 0.7);
2 EnvGen.kr(~envFunc, doneAction: Done.freeSelf);
```

EnvGen example

```
1 ~envFunc = Env.perc(0.1, 0.7);
2 EnvGen.kr(~envFunc, doneAction: Done.freeSelf);
```

Making a note

```
(
{
    ~envFunc = Env.perc(0.1, 0.7);
    ~env = EnvGen.kr(~envFunc, doneAction: Done.freeSelf);
    SinOsc.ar(440) * ~env;
}.play
)
```

Step 7

Output channels & Panning

Spatialized Audio!? 🎧

Channel / Bus

Generally, most generic computers has two output channels:

the **left channel (Bus 0)** &

the **right channel (Bus 1)**.

```
 // to see the channels
s.meter

// Output a synth
Out.ar(bus:0, mySynth);
Out.ar(bus:1, mySynth);
```

Channel / Bus

Generally, most generic computers has two output channels:

the **left channel (Bus 0)** &

the **right channel (Bus 1)**.

```
 // to see the channels
s.meter

// Output a synth
Out.ar(bus:0, mySynth);
Out.ar(bus:1, mySynth);
```

Why do we need the `Out` function?

`Out` function

The `Out` function is used to output the sound to a specific channel.

```
Out.ar(bus:0, mySynth);
```

outputs the synth `mySynth` to the left channel.

`Out` function

The `Out` function is used to output the sound to a specific channel.

```
Out.ar(bus:0, mySynth);
```

outputs the synth `mySynth` to the left channel.

This is particularly useful for multi-speaker setup!

Multichannel Expansion

We can create a function with multiple audio channels by using the array syntax.

```
1 // one channel
2 { Blip.ar(800,4,0.1) }.play;
3
4 // two channels in an array
5 { [ Blip.ar(800,4,0.1), WhiteNoise.ar(0.1) ] }.play;
6
7 // two frequencies in an array
8 { SinOsc.ar([440, 880], 0, 0.2) }.play;
```

Multichannel Expansion + `Out`

We can make the `Out` function output to multiple channels using *multichannel expansion*.

```
(
{
    ~sound1 = Blip.ar(800,4,0.1);
    ~sound2 = WhiteNoise.ar(0.1);
    // the explicit way
    Out.ar([0,1], [~sound1, ~sound2]);

    // using multichannel expansion
    Out.ar(0, [~sound1, ~sound2]);
}.play;
)
```

Multichannel Expansion + `Out`

We can make the `Out` function output to multiple channels using *multichannel expansion*.

```
(
{
    ~sound1 = Blip.ar(800,4,0.1);
    ~sound2 = WhiteNoise.ar(0.1);
    // the explicit way
    Out.ar([0,1], [~sound1, ~sound2]);

    // using multichannel expansion
    Out.ar(0, [~sound1, ~sound2]);
}.play;
)
```

Multichannel Expansion + Out

```
// using multichannel expansion
Out.ar(0, [~sound1, ~sound2]);
```

All sounds are expanded from channel 0 to the next channels.

Multichannel Expansion + Out

```
// using multichannel expansion
Out.ar(0, [~sound1, ~sound2]);
```

All sounds are expanded from channel 0 to the next channels.

What if you have more sound functions than channels?

Multichannel Expansion + Out

```
// using multichannel expansion
Out.ar(0, [~sound1, ~sound2]);
```

All sounds are expanded from channel 0 to the next channels.

What if you have more sound functions than channels?

The sound(s) on unavailable channel(s) will not be played!

The Pan2 function

```
Pan2.ar(in, pos: 0.0, level: 1.0)
```

where `in` is the output sound,

`pos` is the relative position to the center

(-1 is left, +1 is right)

A complete example

```
// Making a note
(
SynthDef(\MyNote, { | freq = 440, amp = 0.5 |
    // Create a wave
    ~wave = SinOsc.ar(freq, 1, amp);

    // Envelope the sound
    ~envFunc = Env.perc(0.1, 0.7);
    ~env = EnvGen.kr(~envFunc, doneAction: Done.freeSelf);

    ~sound = ~wave * ~env;

    // Equally panned on both channels
    Out.ar(0, Pan2.ar(~sound, 0));
}).add
)
```

A complete example

```
// Making a note
(
SynthDef(\MyNote, { | freq = 440, amp = 0.5 |
    // Create a wave
    ~wave = SinOsc.ar(freq, 1, amp);

    // Envelope the sound
    ~envFunc = Env.perc(0.1, 0.7);
    ~env = EnvGen.kr(~envFunc, doneAction: Done.freeSelf);

    ~sound = ~wave * ~env;

    // Equally panned on both channels
    Out.ar(0, Pan2.ar(~sound, 0));
}).add
)
```

[tangent] Spatialized Audio

To achieve spatialized audio, we need more than just panning with `Pan2`.

[tangent] Spatialized Audio

To achieve spatialized audio, we need more than just panning with `Pan2`.

Ambisonics is a technique to achieve spatialized audio.

It uses concepts such as *W, X, Y, Z* to represent the sound in a 3D space.

[tangent] Spatialized Audio

To achieve spatialized audio, we need more than just panning with `Pan2`.

Ambisonics is a technique to achieve spatialized audio.

It uses concepts such as *W, X, Y, Z* to represent the sound in a 3D space.

SuperCollider Ambisonic Toolkit:

doc.sccode.org/Guides/Intro-to-the-ATK.html

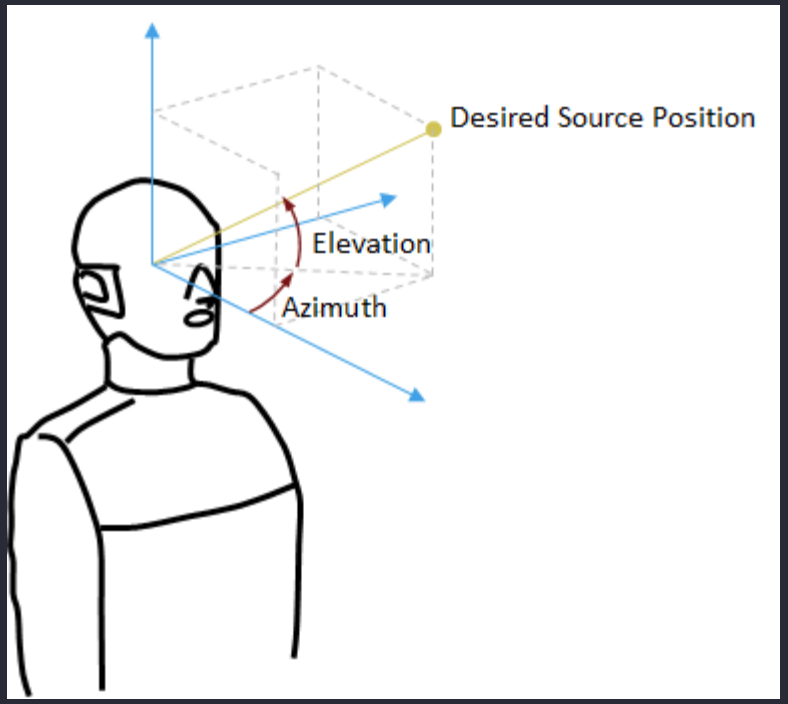
[tangent] Spatialized Audio

[tangent] Spatialized Audio

Head-related transfer function (HRTF) is another technique to achieve spatialized audio.

[tangent] Spatialized Audio

Head-related transfer function (HRTF) is another technique to achieve spatialized audio.



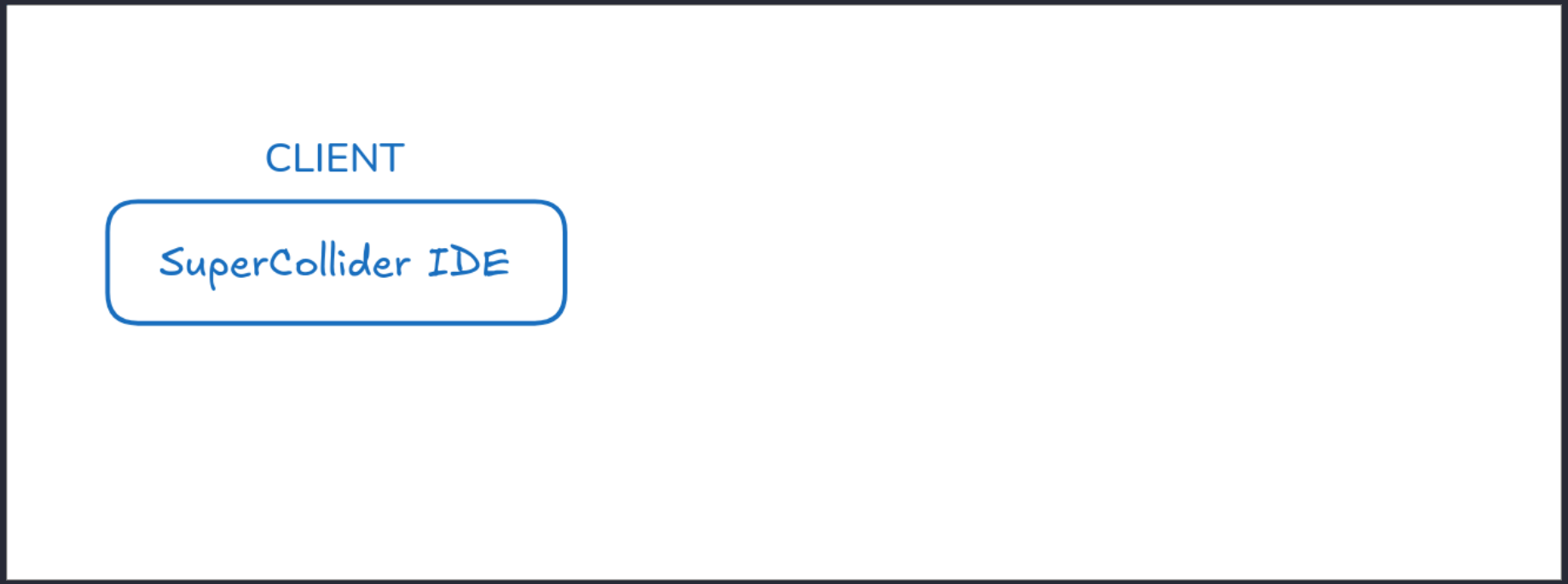
Step 8

OSC message & `scsynth` Server

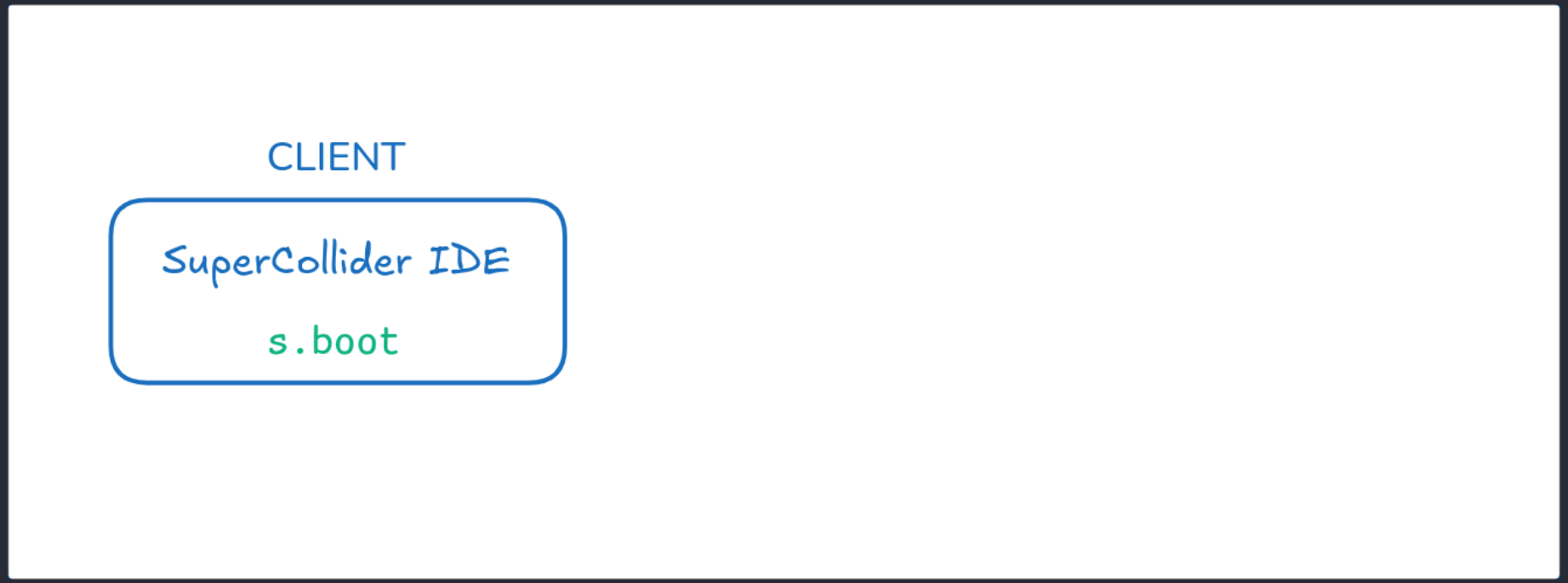
Brace yourself!

The client-server architecture

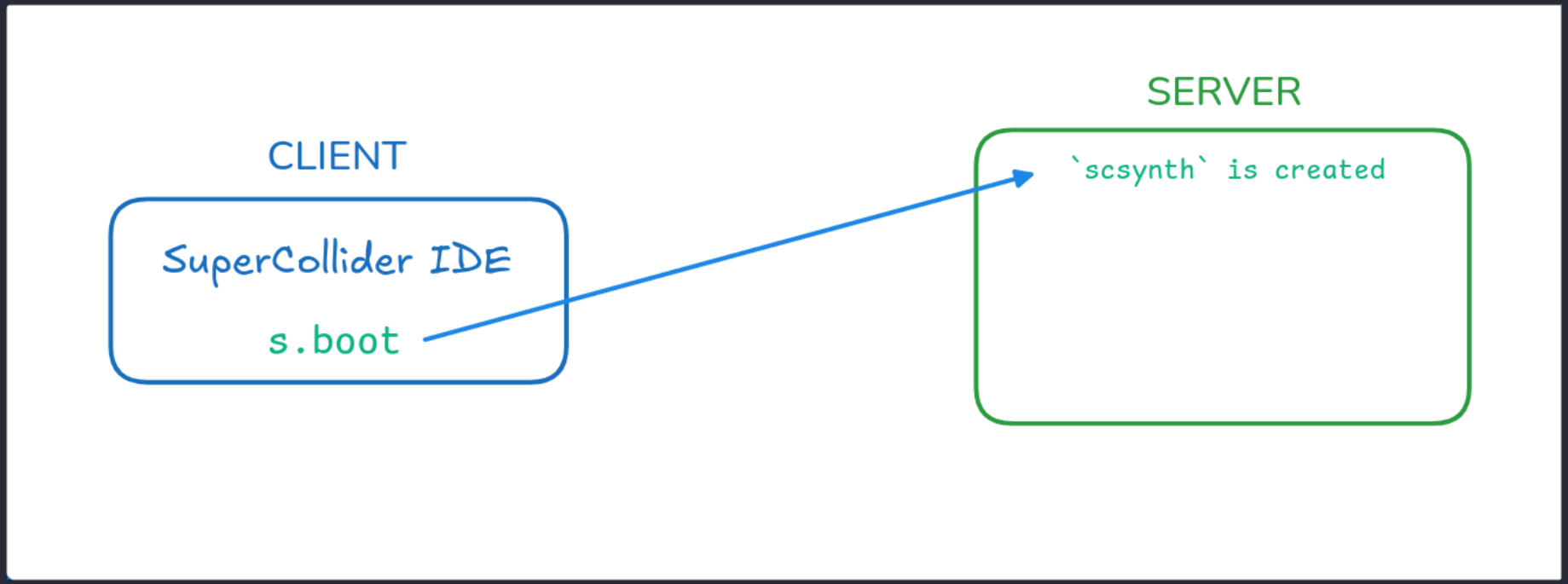
The client-server architecture



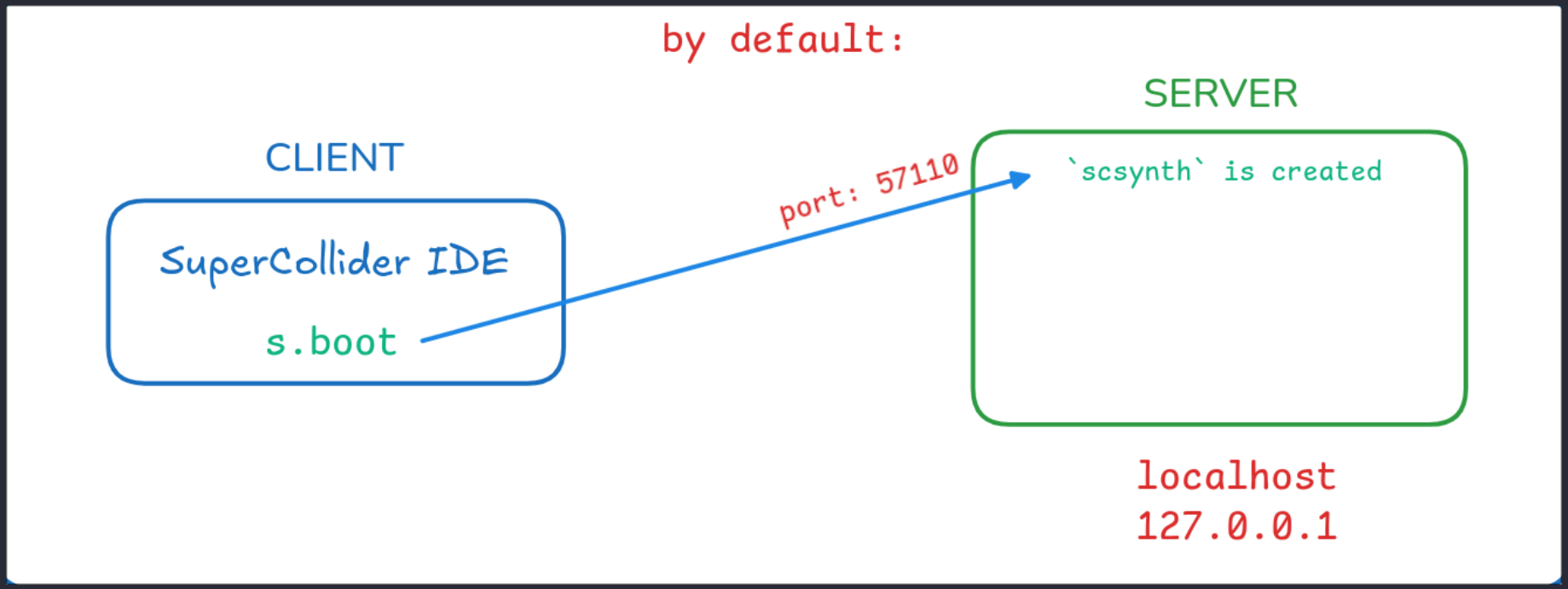
The client-server architecture



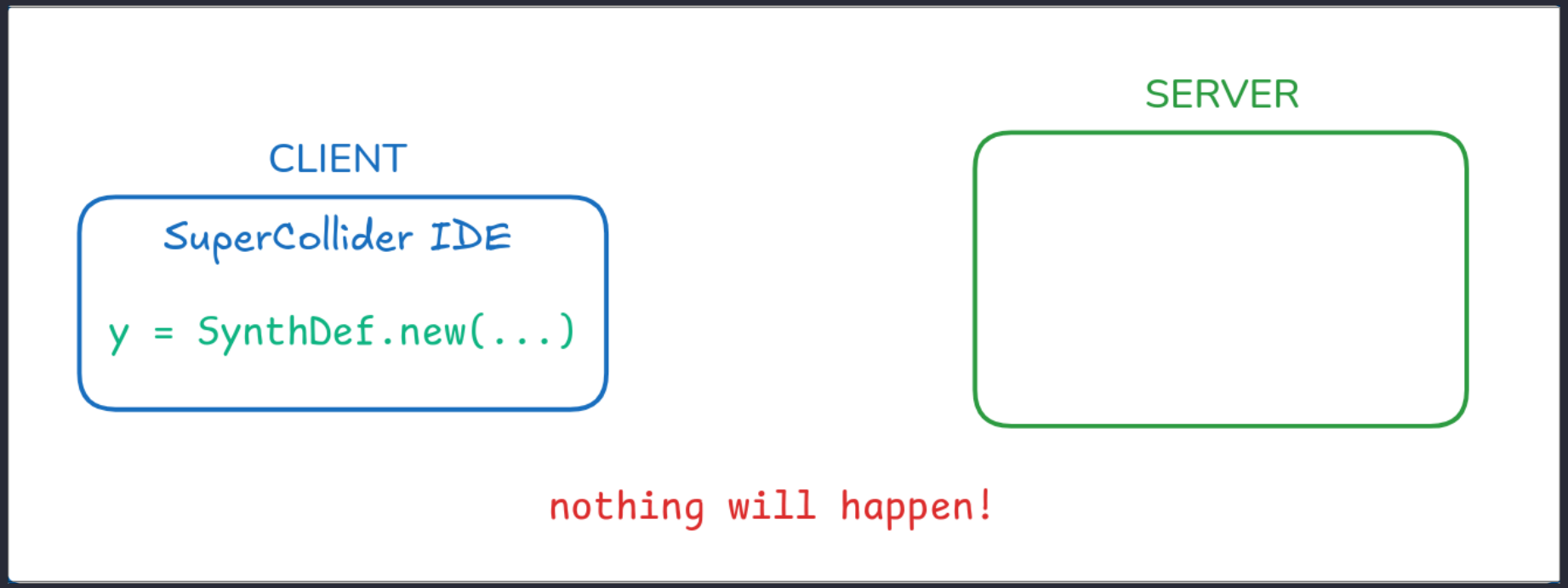
The client-server architecture



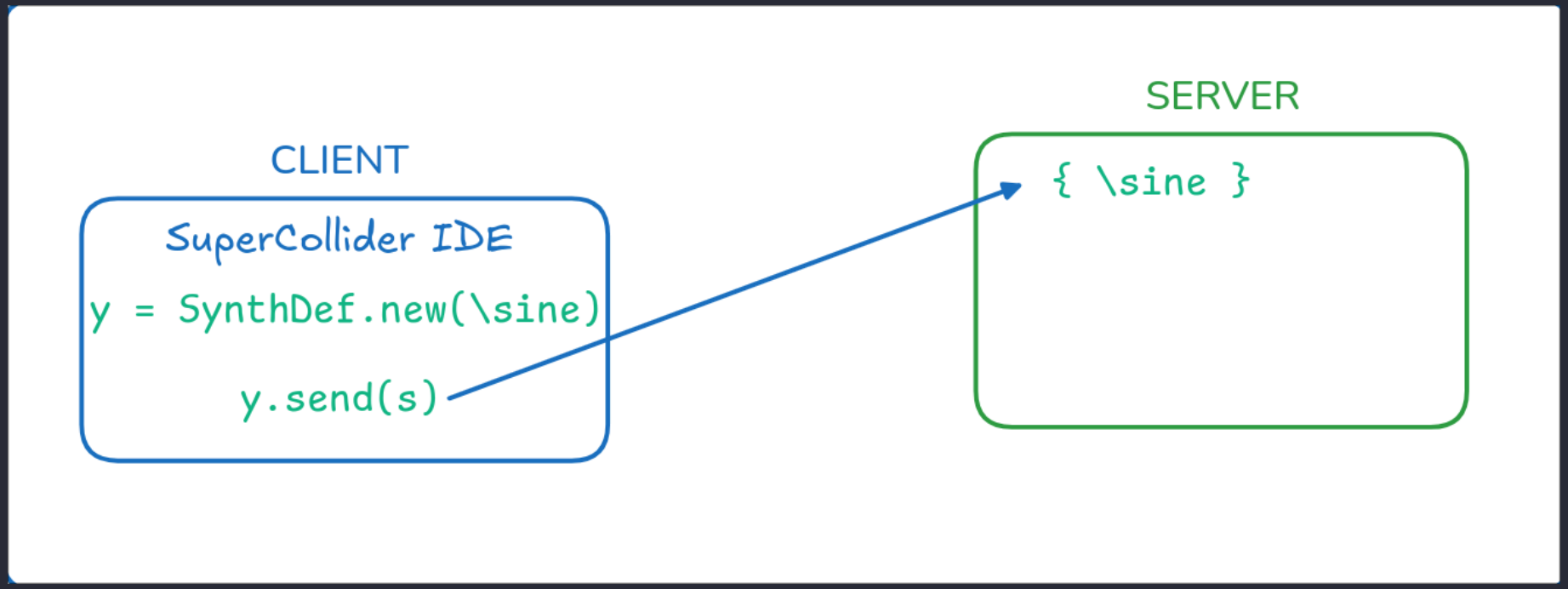
The client-server architecture



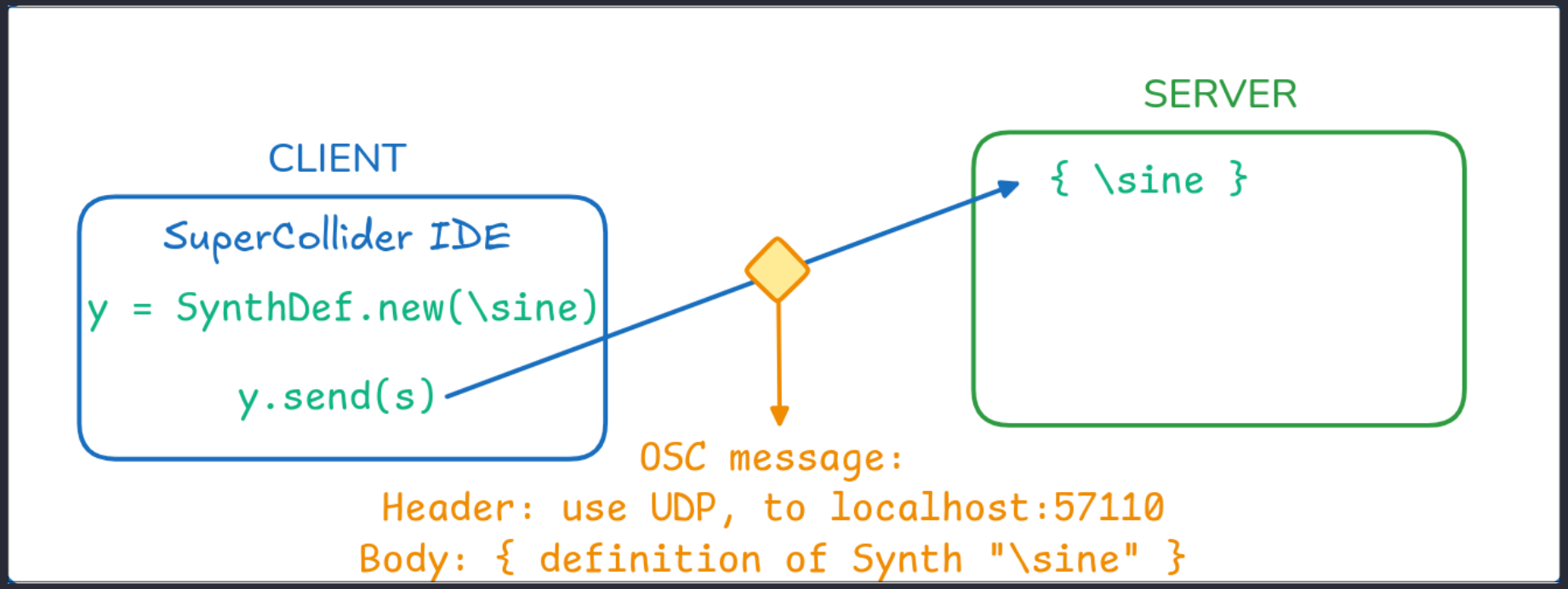
The client-server architecture



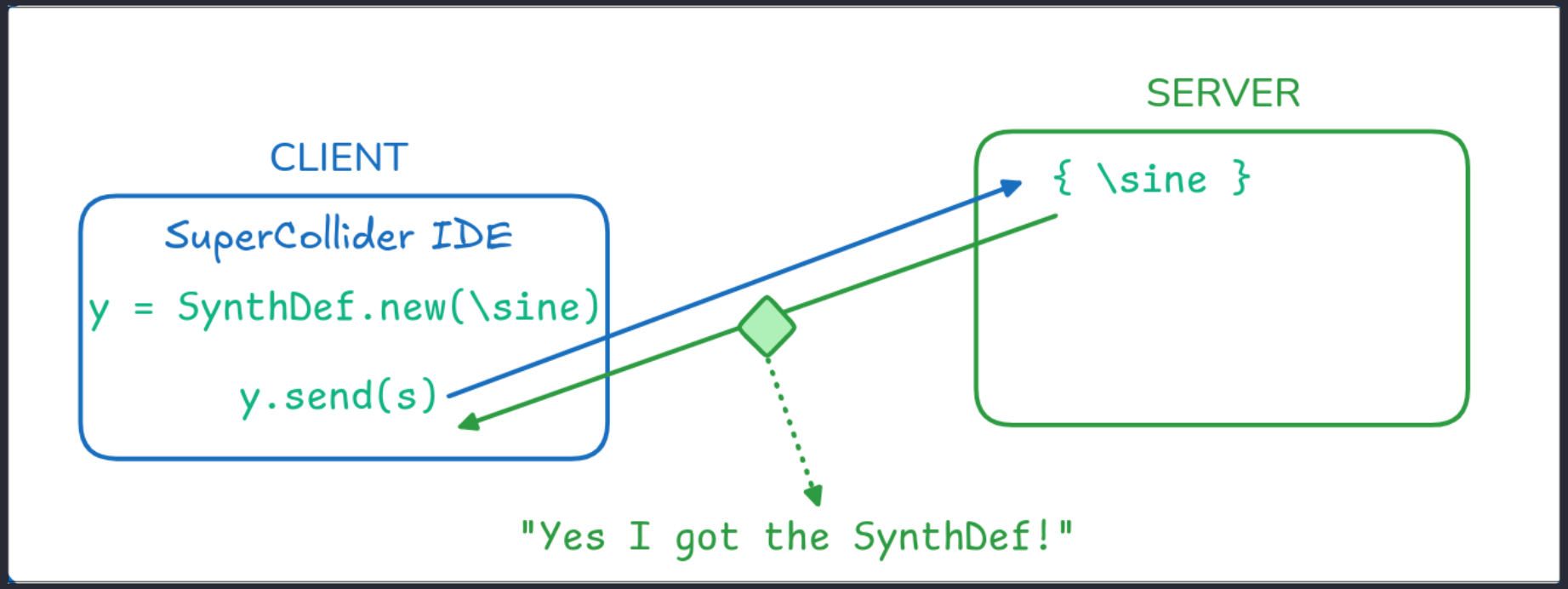
The client-server architecture



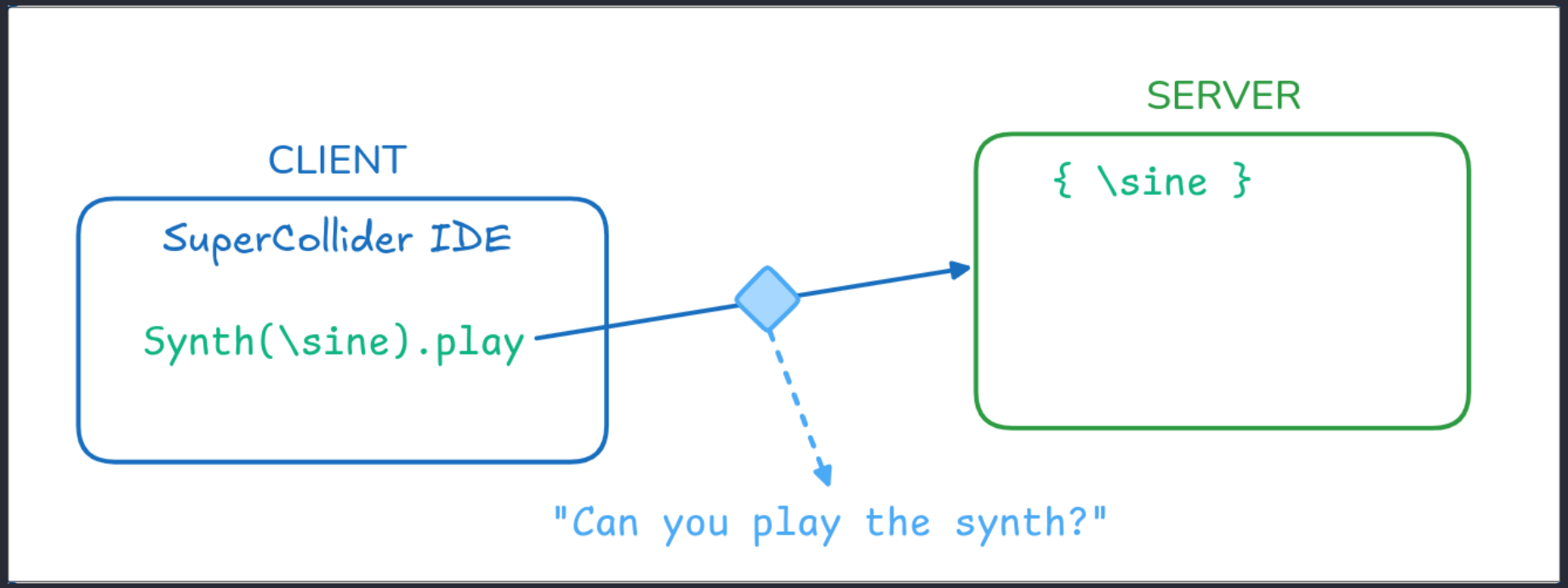
The client-server architecture



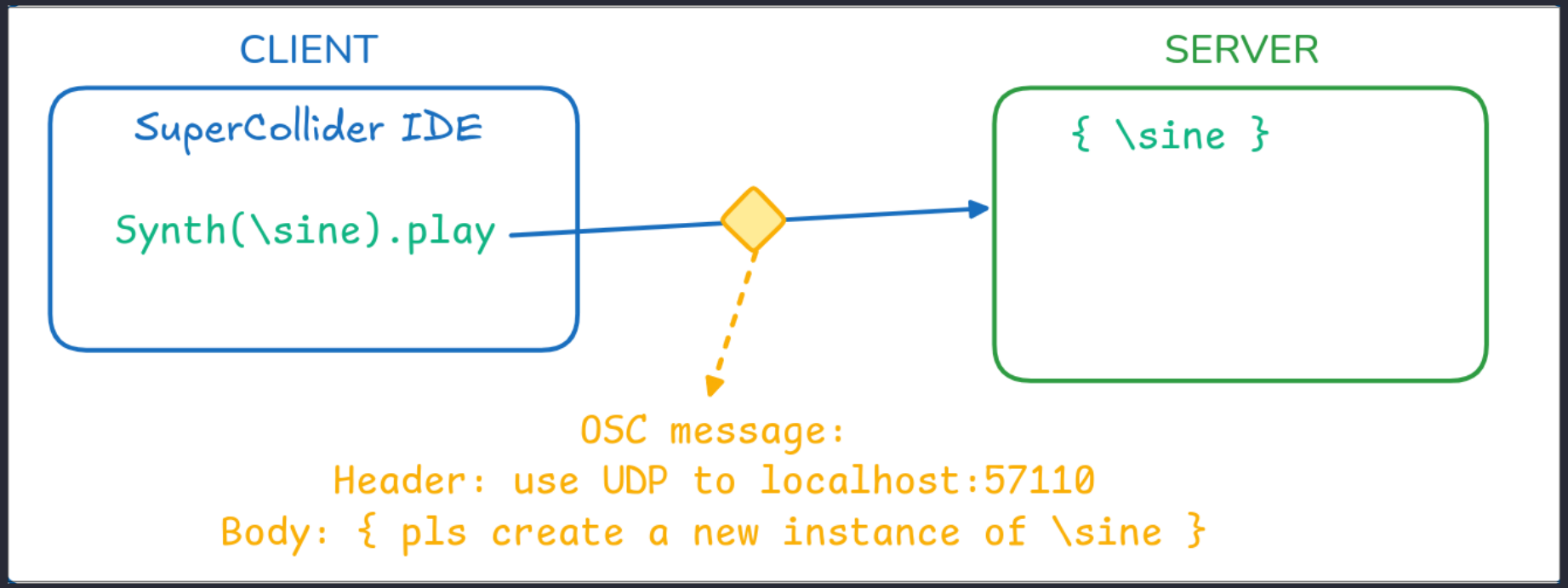
The client-server architecture



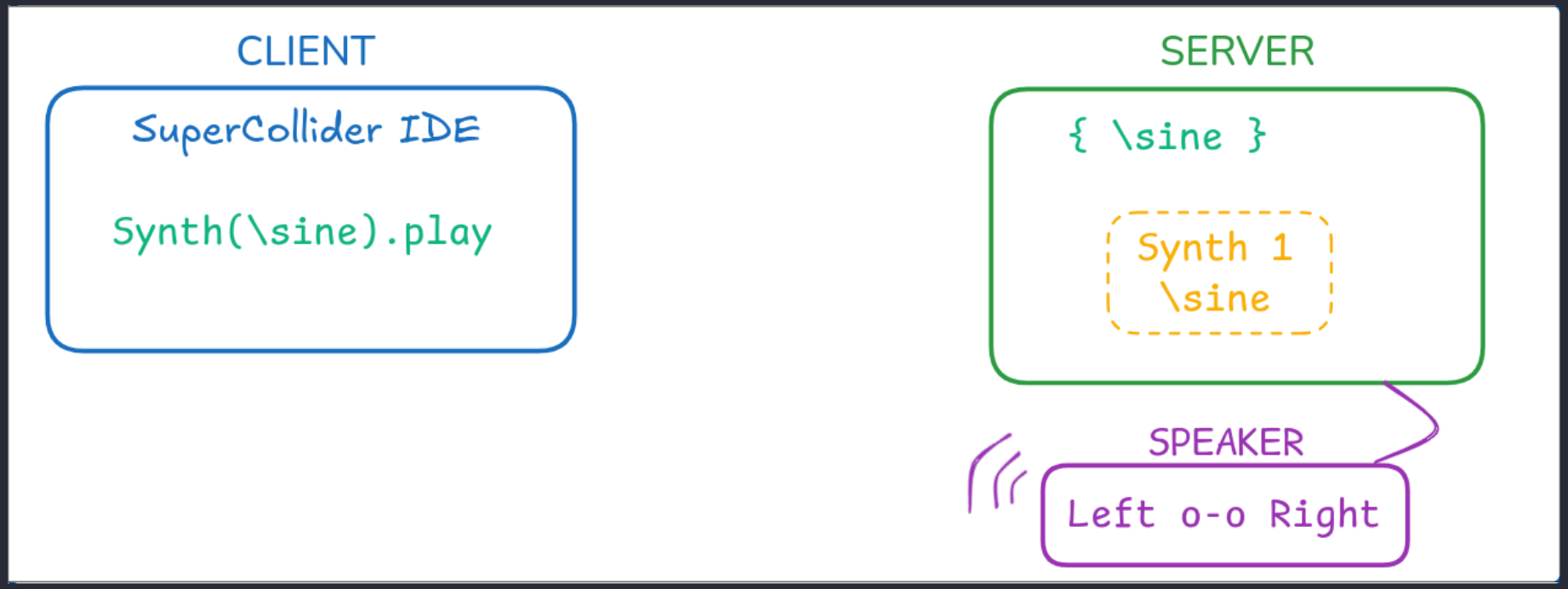
The client-server architecture



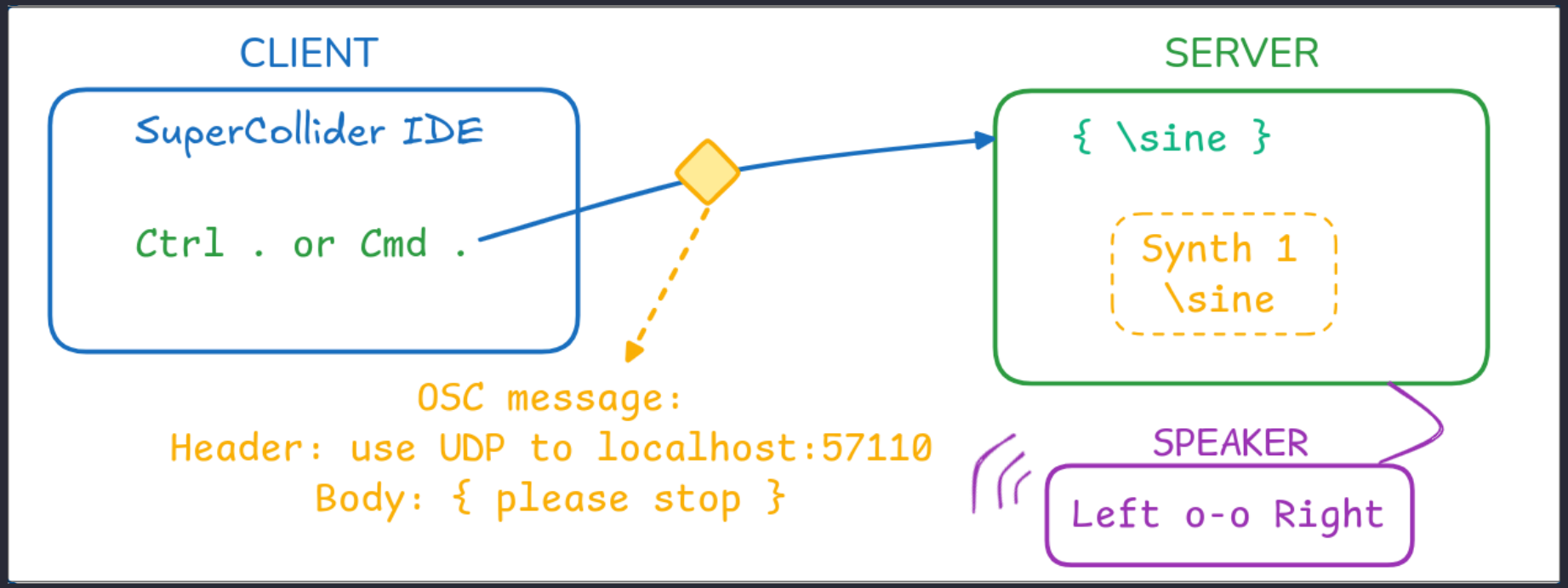
The client-server architecture



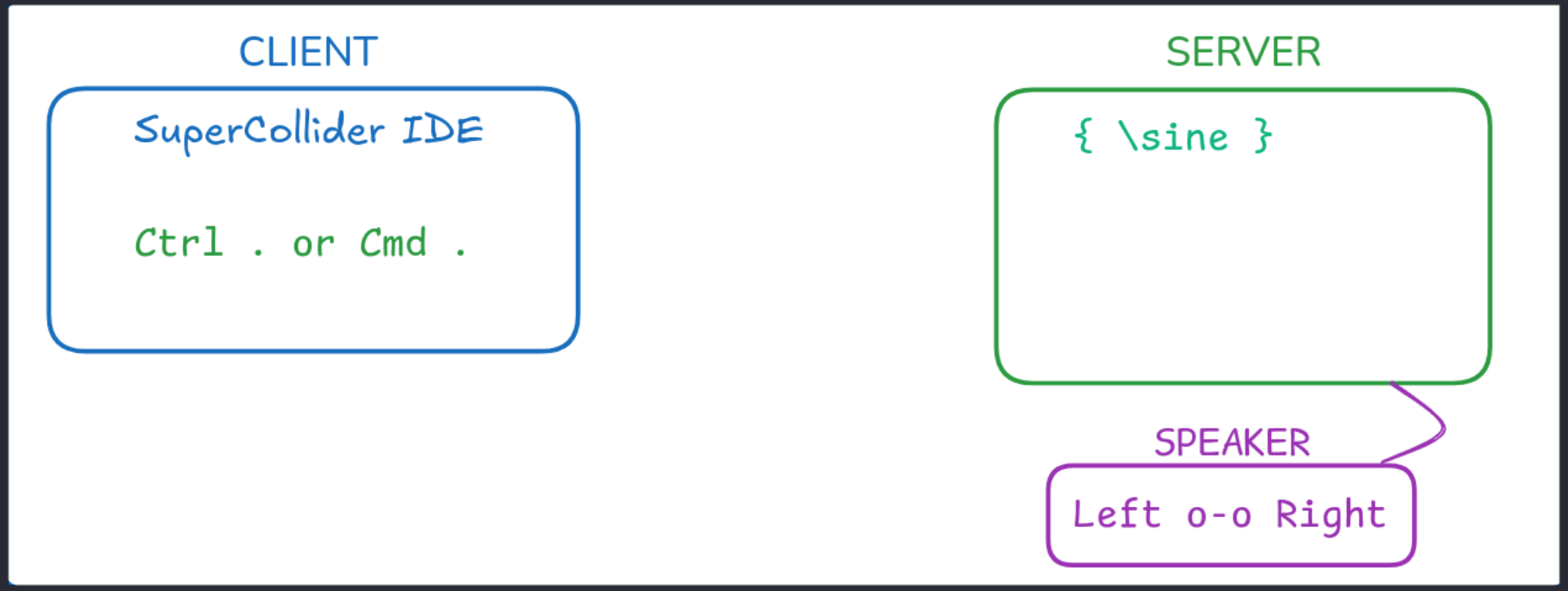
The client-server architecture



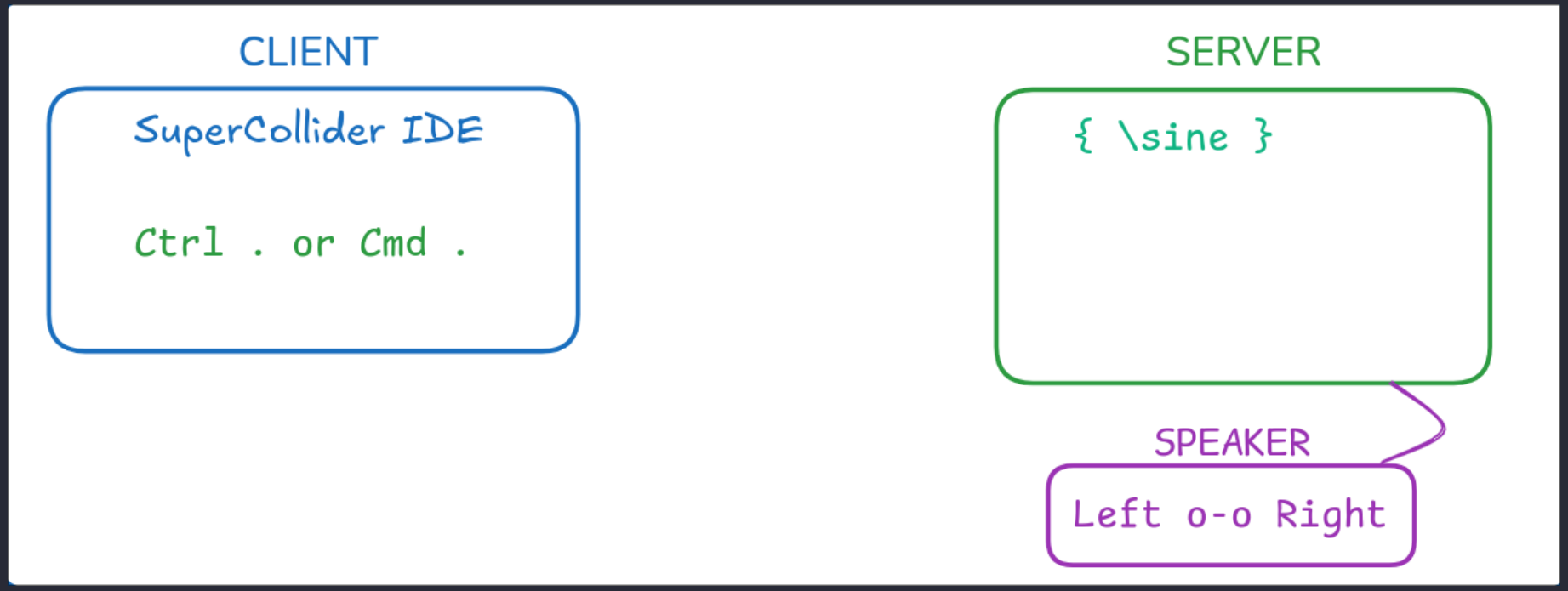
The client-server architecture



The client-server architecture



The client-server architecture



Note: the client is called `sclang`.

How can you interact with the server?

How can you interact with the server?

How can you encode an OSC message?

How can you interact with the server?

How can you encode an OSC message?

What can you do with all of these knowledge?

The `scsynth` server

The `scsynth` server is the server that runs the SuperCollider synthesis engine.

The `scsynth` server

The `scsynth` server is the server that runs the SuperCollider synthesis engine.

By default, it runs on `localhost:57110`

The `scsynth` server

The `scsynth` server is the server that runs the SuperCollider synthesis engine.

By default, it runs on `localhost:57110`

SuperCollider IDE default server variable is `s`.

`s.addr;` reveals the local address

Interact with the `scsynth` server

Interact with the `scsynth` server

```
// Boot the server
s.boot;

// Show the server address & port
s.addr;

// Quit the server
s.quit;

// Show the server output scope (time vs amplitude)
s.scope;

// Show the server input & output meter
s.meter;
```

OSC encoding schema

OSC follows a schema as follows:

```
/address, [values]
/address, [tags, arguments]
```

OSC encoding schema

OSC follows a schema as follows:

```
/address, [values]
/address, [tags, arguments]
```

Example: to create a new synth with the name "sine"

```
/s_new, ["sine"]
```

On the SuperCollider IDE

The IDE provides client-side code that allows user to send OSC messages

On the SuperCollider IDE

The IDE provides client-side code that allows user to send OSC messages

---

By default, it uses the `s` server (`localhost:57110`)

On the SuperCollider IDE

The IDE provides client-side code that allows user to send OSC messages

By default, it uses the `s` server (`localhost:57110`)

```
1 ~mySine = SynthDef("sine", { ... }).add;
2
3 // is equivalent to
4 ~mySine = SynthDef("sine", { ... });
5 ~mySine.send(s);
6
7 // write SynthDef to disk. .send(s) doesn't write to disk
8 ~mySine = SynthDef("sine", { ... }).load(s);
```

On the SuperCollider IDE

The IDE provides client-side code that allows user to send OSC messages

By default, it uses the `s` server (`localhost:57110`)

```
1 ~mySine = SynthDef("sine", { ... }).add;
2
3 // is equivalent to
4 ~mySine = SynthDef("sine", { ... });
5 ~mySine.send(s);
6
7 // write SynthDef to disk. .send(s) doesn't write to disk
8 ~mySine = SynthDef("sine", { ... }).load(s);
```

On the SuperCollider IDE

To play a Synth:

```
1  // play a Synth "\sine" with "freq" augument 900
2  // x is used to keep track of the node number
3  s.sendMsg("/s_new", "sine",
4          x = s.nextNodeID, 1, 1, "freq", 900);
5
6  // to change the frequency of the Synth
7  s.sendMsg("/n_set", x, "freq", 400);
8
9  // to free the node
10 s.sendMsg("/n_free", x);
```

Usage of OSC outside of SuperCollider

OSC allows you to control the synthesizer server from a **different program**, or even a **different machine**!

Usage of OSC outside of SuperCollider

OSC allows you to control the synthesizer server from a **different program**, or even a **different machine**!

This includes but not limited to:

- Python: `python-osc`
- JavaScript: `osc-js`, `osc`

Usage of OSC outside of SuperCollider

OSC allows you to control the synthesizer server from a **different program**, or even a **different machine**!

This includes but not limited to:

- Python: `python-osc`
- JavaScript: `osc-js`, `osc`

We will use `python-osc` for our game later!

Step 9

Extras

Step 9

Extras

Just a brief introduction to the *endless possibility* you can do with SuperCollider.

sclang

`sclang` is the SuperCollider language interpreter. It behaves similarly to Python.

sclang

`sclang` is the SuperCollider language interpreter. It behaves similarly to Python.

You can run *.scd* files using `sclang` on the terminal.

```
sclang mysynth.scd
```

Load different SuperCollider files

You can also load multiple different files with SynthDefs.

Load different SuperCollider files

You can also load multiple different files with SynthDefs.

```
"mySynth1.scd".load;
"mySynth2.scd".load;
```

Loop, repeat, & Routine

You can set up repeated tasks in SuperCollider using a `n.do()`, a `Routine`, `.loop`, or `.repeat()` methods.

```
20.do({ |i|
    i.postln; // print all numbers from 0 to 19
});


[1, 5, 3].do({ |i|
  i.postln;  // print out 1, 5, 3
});


t = Task({ { "Yippee".postln; 1.wait; }.loop});
t.start;
t.stop;
```

`wait`

You can set up a time pause between events or tasks using the `.wait` method inside a `Task` or a `Routine`.

wait

```
t = Task({
    {
        "Yippee".postln;
        1.wait; // wait for 1 second
    }.loop});
t.start;
t.stop;

x = Routine(
  10.do({
    "Howdy".postln;
    1.wait; // wait for 1 second
    });
)

x.play();
```

MIDI

SuperCollider can also interact with MIDI devices, uses MIDI standards, etc.

```
~E4midi = 64.midicps; // convert MIDI note to frequency

// Example: Play a MIDI note
{ SinOsc.ar(64.midicps); }.play();

MIDIClient.init; // Start the MIDI client
MIDIIn
MIDIOut
MIDIFunc
```

Application

Application

Let's make a game!

Game time! 🎮

You will implement a Python Turtle game that uses OSC to synthetize sound effects.

Link to the game template:
https://github.com/notkaramel/RunningBlobby

```
python --version    # to check if you have Python installed
```

Set up the game repository:

```
git clone https://github.com/notkaramel/RunningBlobby
cd RunningBlobby

python3 -m venv .venv
source .venv/bin/activate

(.venv) pip install -r requirements.txt

# to run the game
(.venv) python main.py
```

GOAL: Create a game that uses OSC and SuperCollider to create *spatialized audio*

Tasks

You will implement a few `SynthDef`s in SuperCollider

- `\walk`: when the player runs
- `\wall`: when the player hits the wall
- `\spin`: when the player spins

All SynthDefs takes *xPosition* and *yPosition* as arguments.

Value range: -1 for most left/top, +1 for most right/bottom

Play the game

Make sure you have a running SuperCollider server at `localhost:57110`.

Run the game from the command line with

```
(.venv) python game.py
```

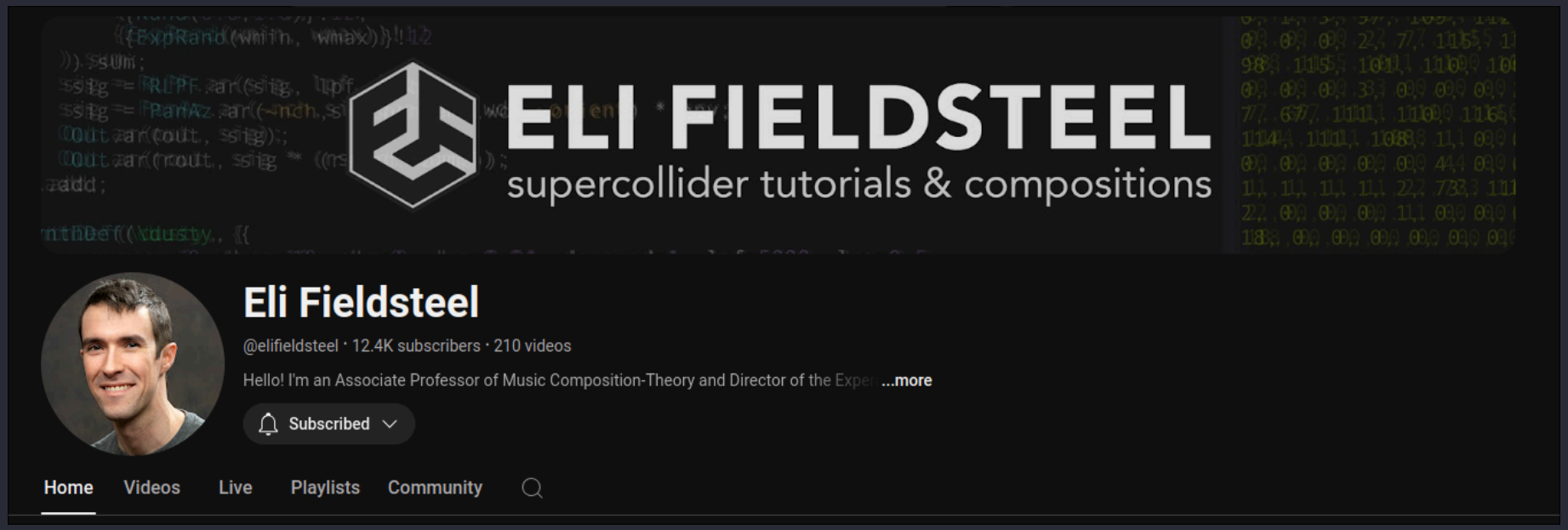
Use arrow keys to move the running blobby.
Press `Space` to spin.

Thank you for attending the workshop!

Thank you for attending the workshop!

Any questions?

Curious to learn more about SuperCollider?



https://www.youtube.com/@elifieldsteel

Curious to learn more about SuperCollider?

Curious to learn more about SuperCollider?

SuperCollider Tutorials

by Eli Fieldsteel

Playlist • 32 videos • 335,599 views

Play all

1 

SuperCollider Tutorial: 0. Introduction

Eli Fieldsteel • 110K views • 11 years ago

2 

SuperCollider Tutorial: 1. Navigating the Environment

Eli Fieldsteel • 103K views • 11 years ago

3 

SuperCollider Tutorial: 2. Making Sound

Eli Fieldsteel • 124K views • 11 years ago

4 

SuperCollider Tutorial: 3. Synth and SynthDef

Eli Fieldsteel • 76K views • 11 years ago

5 

SuperCollider Tutorial: 4. Envelopes and doneAction

Eli Fieldsteel • 51K views • 11 years ago